NAIC-ID(RS)T-0022-95

NATIONAL AIR INTELLIGENCE CENTER

SOFTWARE TRANSPLANTATION AND UNDERSTANDING TOOL--THE STUDY AND REALIZATION OF THE VAX-C DECOMPILE SYSTEM

by

Hou Wenyong, Lu Jiquan, et al.

HUMAN TRANSLATION

NAIC-ID(RS)T-0022-95   25 May 1995

MICROFICHE NR: 95C000328

SOFTWARE TRANSPLANTATION AND UNDERSTANDING TOOL--THE STUDY AND REALIZATION OF THE VAX-C DECOMPILE SYSTEM

By: Hou Wenyong, Lu Jiquan, et al.

English pages: 12

Source: Jisuanji Gongcheng, Vol. 18, Nr. 3, 1992; pp. 1-4

Country of origin: China
Translated by: SCITRAN
F33657-84-D-0165
Requester: NAIC/TATA/Keith D. Anthony
Approved for public release: distribution unlimited.

THIS TRANSLATION IS A RENDITION OF THE ORIGINAL FOREIGN TEXT WITHOUT ANY ANALYTICAL OR EDITORIAL COMMENT STATEMENTS OR THEORIES ADVOCATED OR IMPLIED ARE THOSE OF THE SOURCE AND DO NOT NECESSARILY REFLECT THE POSITION OR OPINION OF THE NATIONAL AIR INTELLIGENCE CENTER.	PREPARED BY: TRANSLATION SERVICES NATIONAL AIR INTELLIGENCE CENTER WPAFB, OHIO

GRAPHICS DISCLAIMER

All figures, graphics, tables, equations, etc. merged into this translation were extracted from the best quality copy available.

Accesion For	
NTIS CRA&I	✓
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and / or Special
A-1	

SOFTWARE TRANSPLANTATION AND UNDERSTANDING TOOL--THE STUDY AND REALIZATION OF THE VAX-C DECOMPILE SYSTEM

/1*

Hou Wenyong Lu Jiquan Shi Shumin Fu Ming

ABSTRACT

The decompiler is a type of tool associated with software understanding and software transplantation as well as secondary development. It is capable of taking low level languages and translating them into high level languages. This article carries out an overall introduction of such areas as general organization and design of decompilers, optimized reduction, control flow analysis, as well as data analysis, and so on. This decompiler system is realized on Micro-VAX II VMS4.4 operating system.

KEY WORDS Software tool, General organization, Optimized reduction, Control flow, Data flow

A decompiler is a type of software tool capable of taking machine code or compilation code programs and translating them into equivalent high level language programs. It is a type of tool associated with the carrying out of software understanding and software transplantation as well as development. It brings with it extremely great advantages for conversion to domestic production of software and the work of signification.

As far as the development of decompilers up to now is concerned, it still has not become a type of mature theory. Currently available reference materials are extremely scarce. As a result, decompiler work at the present time is still empirical and exploratory. Research on a type of mature decompiler theory has very great significance for such operations as software understanding and development.

* Numbers in margins indicate foreign pagination. Commas in numbers indicate decimals.

The earliest reported decompiler work is a series of software tools developed by U.S. electronics laboratories in the period 1960-1963. After 10 years, in 1973, Hollarder [1], in his doctoral dissertation, introduced taking IBM 360 compilation subsets and translating them into decompilation devices associated with ALGOL language. The beginning of Chinese decompilation research was relatively late. In it, the Hefei Industrial College developed a 68000 C decompilation device. In Dual 68000 systems, C language programs can be directly run after relatively simple C programs go through decompilation. It requires relatively large amounts of artificial intervention. In this article, we will introduce a C compilation system associated with VAX/VMS4.4 operating systems. The technology which it makes use of possesses special functions. It assimilates the good points of other decompilers. However, it possesses its own characteristics.

/2

1 C DECOMPILATION SYSTEM GENERAL ORGANIZATION

C decompilation systems are decompilation devices realized using C language with VMS operating systems associated with Micro VAX II. Their objective is to take target operating code and translate it into high level C language programs. VAX machines are a type of relatively popular minicomputer system. They belong to a series of optimized Chinese computers. On the machines in question, the realization of C decompilation will provide powerful tools for conversion to domestic software production.

In the process of decompilation operations, it is necessary to carry out repeated scanning of code and analyze it step by step. In the intervals, there will be produced multiple layers of intermediate results. Normally, these appear as a type of language higher than compilation but also not reaching the form of high level C language programs, that is, a type of transitional intermediate language. In order to display this type of intermediate result, in conjunction with making it appropriate to translations associated with the intervals between various layers of intermediate results, we designed a type of program internal memory storage form, satisfying the requirements associated with expressing intermediate results. We designate this data organization form as program graphic form.

The C decompilation system in question is based on the characteristics of C language. It carries out organizational analysis and recovery, maintaining the characteristics of C language. It includes a decompilation software and detranslation software. Due to the fact that decompilation is a stand alone software, it belongs to a different topic. This article focuses on introducing several aspects of design in detranslation. General system organization is as shown in Fig.1.

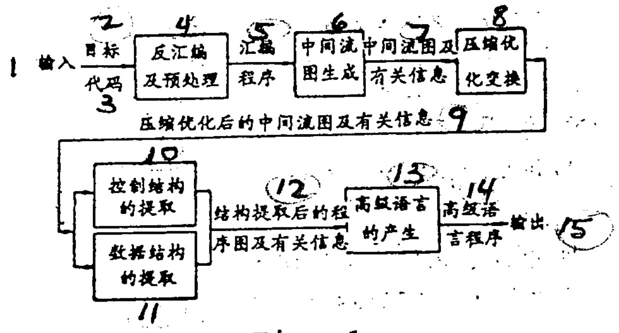


Fig. 1

Key: (1) Input (2) Target (3) Code (4) Decompilation and Preprocessing (5) Compilation Program (6) Generation of Intermediate Flow Graph (7) Intermediate Flow Graph and Related Information (8) Reduction Optimized Translation (9) Post Reduction Optimized Intermediate Flow Graph and Related Information (10) Control Structure Recovery (11) Data Structure Recovery (12) Post Structure Recovery Program Graphs and Related Information (13) Production of High Level Language (14) High Level Language Program (15) Output

2 VARIOUS DECOMPILATIONS AND PREPROCESSING

Decompilation is an independent software. It acts as a detranslation preprocessing section. Decompilation here not only considers the universal usefulness. It also considers useful supplementary information supplied for detranslation in order to facilitate data flow analysis and control flow analysis. In compilation code, compilation commands associated with complex functions are included. With regard to these commands, it is necessary to carry out analysis and optimization. Using basic commands, compilation command sequences display a complex operation. This type of preprocessing is necessary. It is possible to simplify data flow analysis and control flow analysis.

3 FORMATION OF INTERMEDIATE FLOW GRAPHS

We designed a type of program graph form to satisfy the requirements associated with the different levels of intermediate result during detranslation analysis processes in order to facilitate flexible realization of transformations between the various levels of intermediate results. Using graphical form to store these results is unusually suitable to internal machine memory. At the same time, during detranslation operation processes, it is necessary to carry out depth prioritized searches on program graphs. Graphical memory storage forms are also very suitable to the realization of this type of search.

Among the basic junction points associated with program graphs, a few keys are introduced below: operation character, operation number 1, operation number 2, operation number 3, operation number 4, output 1, output 2. Meanings are as follows:

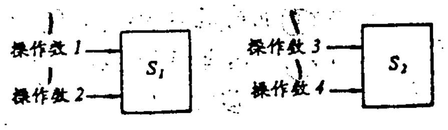
(1) operation number 3: = operation number 1 [operation character] operation number 2

When operation characters are not conditional directional operations, output 2 indicates a unique follow on associated with the junction point in question. When operation characters are conditional and directional, output 1 indicates a follow on associated with the condition being true. Output 2 is a follow on where the condition is false.

(2) If operation characters stand for a sentence pattern in high level C language, operation numbers also have a new function. Here, an individual basic nodal point represents nothing else than a statement in high level language, using this individual basic nodal point to act as the root of a tree. For example, if-else statement

if (Logic-exp)S_1 else S_2;

operation character: "if" output 1: indicates Logic-exp
output 2: indicates the next follow on nodal point



Key: (1) Operation Number

In order to store program graphs, a space ins was opened. At the same time as forming program graphs, we also set up process information blocks, various types of variable tables, various types of constant tables, address marker trees, and other similar related information. These data combine together forming the general data storage space associated with detranslation systems.

4 REDUCTION AND OPTIMIZATION TRANSLATIONS ASSOCIATED WITH INTERMEDIATE FLOW GRAPHS

Compression optimization translations are for the purpose of reducing as much as possible detranslation result code expansion problems. They take a number of data operations in original /3 compilation programs and transmit commands in accordance with processes associated with fixed condition combined transformations in order to raise the readability of detranslation outputs, causing detranslation results to possess even more the style of high level languages. At the same time, they also avoid a number of unstructured problems.

For example, there is a C statement

```
if (a= =0) && (c=(X x 2-8)<5))
{Block 1};
else {Block 2;  }
```

The corresponding compilation program flow graph is as in Fig.2.

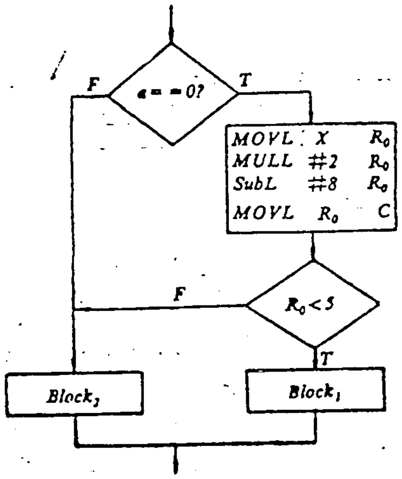


Fig. 2

This is an unstructured flow chart associated with "abnormal selection paths". If direct translation is made in accordance with this flow chart into C language, it will cause programs to become difficult to understand. However, going through optimized reduction translations, we are then able to obtain the structured program flow chart which can be induced as shown in Fig.3.

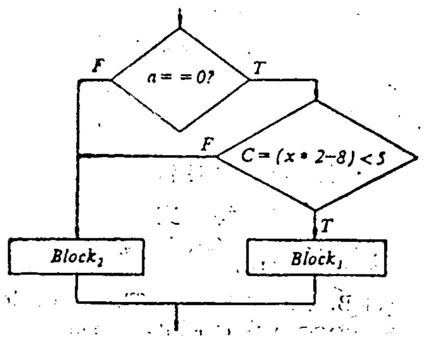


Fig. 3

Optimized reduction translations are relatively complicated. will introduce them in another article.

5 SYSTEM CODE IDENTIFICATION AND FUNCTION INFORMATION STORAGE

Generally speaking, among target codes which it is possible to run, in all cases, they exist in system codes. This includes two areas: one is system start up codes; two is system functions. These codes are inserted into target codes. Decompilation must furnish special processing.

In a VAX-C operating environment, system code is transferred to operating program storage. Among target codes which it is possible to run, only a single entry address associated with the operating code in question within operating storage is provided. In conjunction with this, storage code has not yet been taken and inserted into target code. Because of this, we are able, on the basis of this entry, to precisely determine what storage code has been transferred. In conjunction with this, it is possible to obtain the corresponding C language storage function name.

To this end, we formed a standard function information storage. The first part of information storage is relational storage corresponding to function entry and function name. During decompilation, if corresponding relationships matching operating codes are determined, then, it is possible to transfer function information storage, directly--on the basis of entry addresses--inquiring about what types of functions they are.

Besides this, during decompilation, when carrying out data flow analysis, there is still a requirement for such information as the reentry type associated with the various functions as well as the nature of the various parameters and so on. Therefore, the second part of function information storage is nothing else than these pieces of information.

6 CONTROL STRUCTURE ANALYSIS AND C CHARACTERISTIC PROCESSING

Recovery of program control structures is nothing else than carrying out analysis of program control flow, searching out basic structural components, and illucidating equivalent high level control structures in terms of semantics.

The first is the recovery of logical forms of expression, that is, the merging of logical forms of expression. Logical forms of expression obtained by merging are selections associated with higher level languages--for example, conditional decision forms associated with cyclical statements. Here, the principal basis is the four rules below (see Fig. on next page).

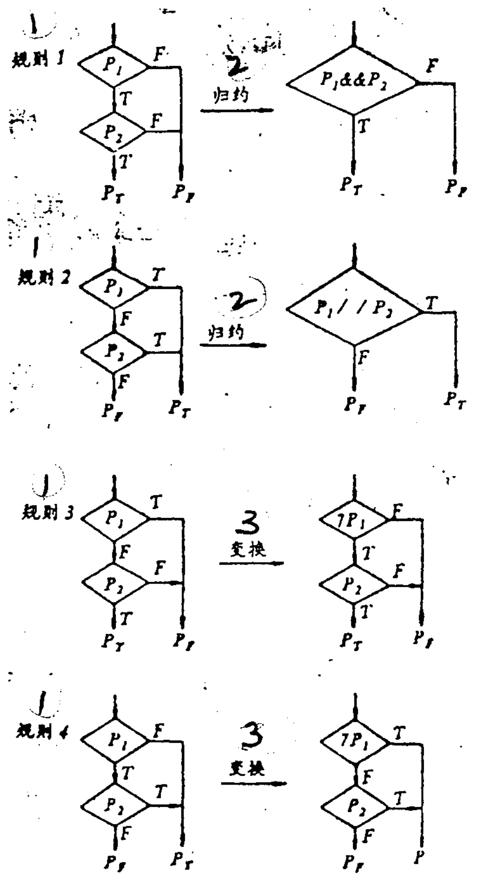
On the basis of logical forms of expression after merging as well as other information, through control flow analysis, it is then possible to induce high level language control structures. We will introduce contents in this area in another article.

This system not only realizes the recovery of general control structures. Moreover, it also restores statement break and Continue statements possessing C language characteristics.

7 C DECOMPILATION DATA STRUCTURE ANALYSIS

In compilation programs, information associated with related data structures is fully and implicitly defined in operating /4 codes and operating numbers. There is a strong relationship to the address search methods associated with compilation programs.

/4



Key: (1) Rule (2) Induction (3) Translation

Data structure recovery is divided into three stages: information restoration, merging, and aggregate analysis.

7.1 Recovery of Information Utilizing Variables

(1) On the basis of type markings in compilation commands

(2) On the basis of parameter type determinations associated with standard C language function storage

(3) On the basis of address search methods for changes between addresses in order to make determinations

(4) On the basis of special commands

7.2 Quality Type Mergers

In compilation programs, one individual variable is capable of having multiple forms of application. Moreover, the types displayed by the various forms are capable of wide divergences. Generally speaking, then, there are the several situations below (only two types of application).

(1) Arrays and structural elements

(2) Simple types and structure types

(3) POI and array type

(4) Full form and guide type

(5) POI and structural elements

(6) With symbol digits and without symbol digits

As far as these types of combinations as well as mergers of various forms of application information are concerned, it is necessary to deal with each case on its own terms. Mergers of two types of application information are quite simple to calculate. Combinations of multiple types of application information, by contrast, must go through complicated comprehensive analysis.

7.3 Comprehensive Analysis of Variable Types

From application information associated with variables, the type associated with consolidating them is a difficult operation.

There are two areas: one is the necessity to carry out combinations when types cited for variables at two points are not the same; the second is the carrying out of consolidations of variable type information already obtained, once again executing processes associated with eliminating the false and storing the true.

Data flow analysis is a complicated process. Its general organization is as shown in Fig.4.

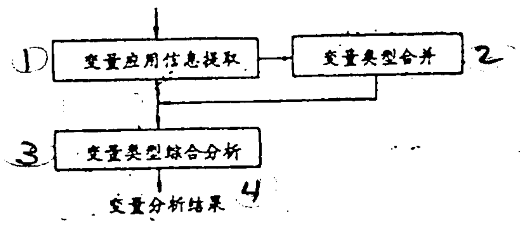


Fig.4

Key: (1) Variable Application Information Recovery
(2) Variable Type Merger (3) Variable Type Comprehensive Analysis
(4) Variable Analysis Results

8 INDETERMINATE PROCESSING ASSOCIATED WITH VARIABLE ANALYSIS

Due to the consolidation of variable types during data flow analysis possessing a definite dual significance, there exists, in variable analysis, therefore, a problem with indeterminance. With regard to this problem, the principle is: maintain the consistency between the equivalency and functions associated with decompilation results and target code.

9 THE PRODUCTION OF HIGH LEVEL C LANGUAGE

The production of high level C language is taking information in intermediate flow graphs and using C language forms for output. It includes the several areas below:

(1) Overall variable output

(2) System predefined variable output

(3) Function parameter as well as local variable and register variable output

(4) Program statement output

REFERENCES

1 Hopwood. Gregory Littell 1978 Decompilation, pH. D dissertation

2 Mosgol B. Tool Ada and The "Middle End' of the PQCC Ada Compiler. 1980

3 唐雅松, 郝克刚. 计算机软件开发方法、工具和环境. 西安: 西北大学出版社

(Editor Diao Lieshen)

DISTRIBUTION LIST

DISTRIBUTION DIRECT TO RECIPIENT

ORGANIZATION	MICROFICHE
B085 DIA/RTS-2FI	1
C509 BALLOC509 BALLISTIC RES LAB	1
C510 R&T LABS/AVEADCOM	1
C513 ARRADCOM	1
C535 AVRADCOM/TSARCOM	1
C539 TRASANA	1
Q592 FSTC	4
Q619 MSIC REDSTONE	1
Q008 NTIC	1
Q043 AFMIC-IS	1
E404 AEDC/DOF	1
E408 AFWL	1
E410 AFDTC/IN	1
E429 SD/IND	1
P005 DOE/ISA/DDI	1
P050 CIA/OCR/ADD/SD	2
1051 AFIT/LDE	1
P090 NSA/CDB	1

Microfiche Nbr: FTD95C000328L
NAIC-ID(RS)T-0022-95